Transitioning From Software Requirements Models to Design Models

PI: Jon Whittle

QSS Group Inc.

NASA POC: Michael Lowry

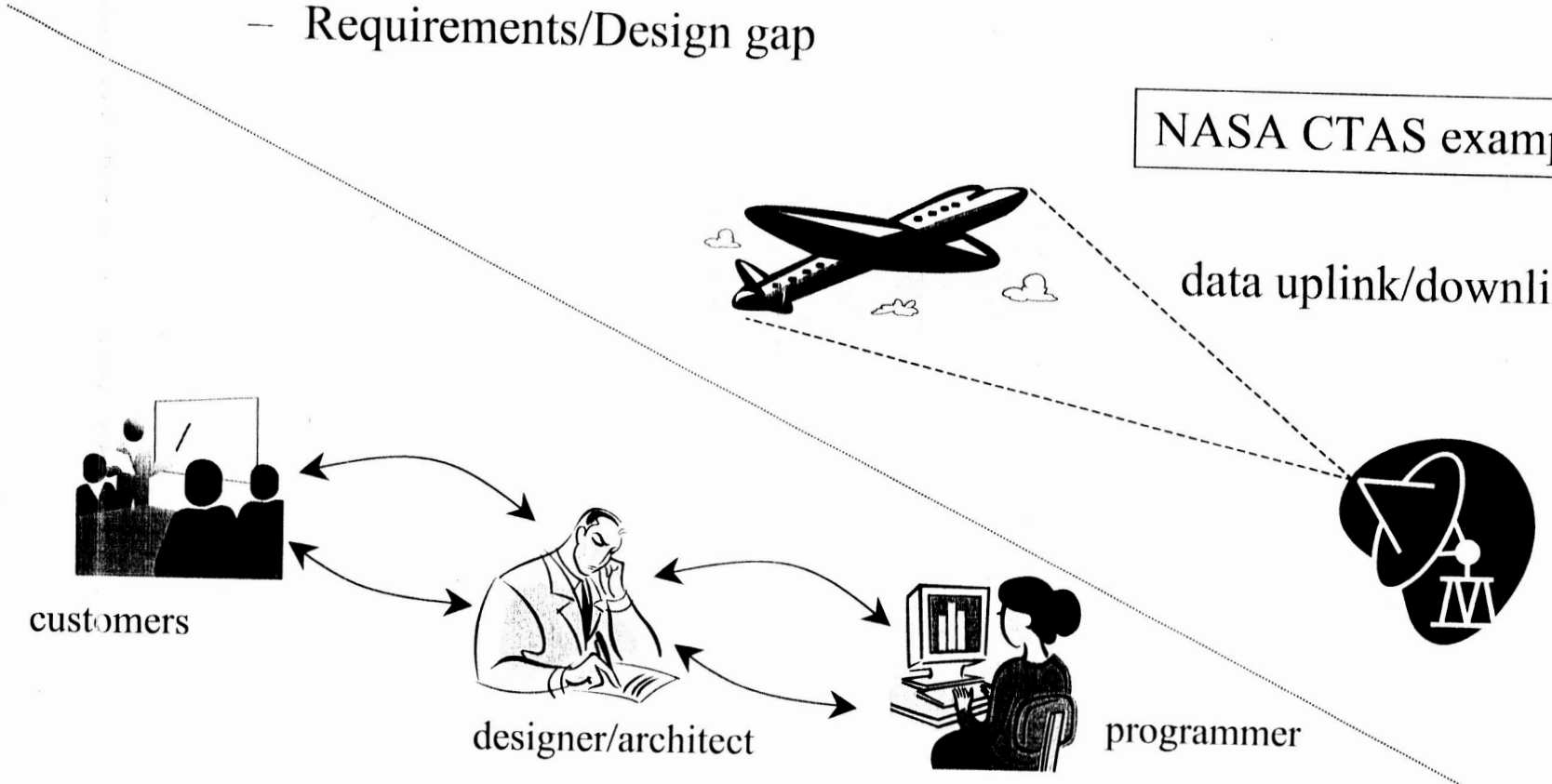
Ames Research Center

Problem

How to cope with the problems of requirements engineering?

- Complexity
- Validation
- Evolution
- Requirements/Design gap

NASA CTAS example:



Information Sciences & Technology
Ames Research Center

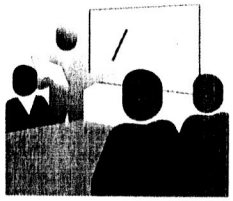
Approach

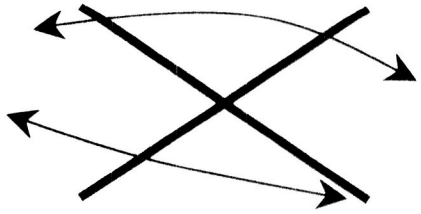
- Complexity
- Validation
- Evolution

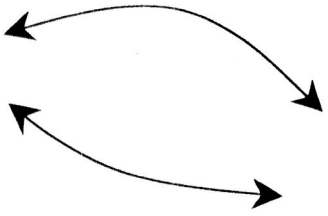
→ Simulation of use cases (scenarios)

- Requirements/Design gap

→ Semi-automatic design transformation





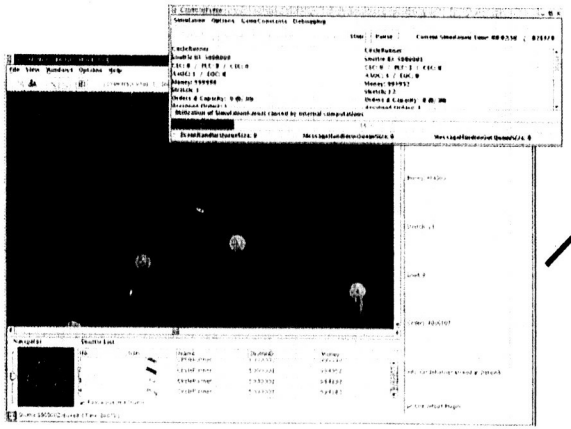


customers

designer/architect

programmer

simulation

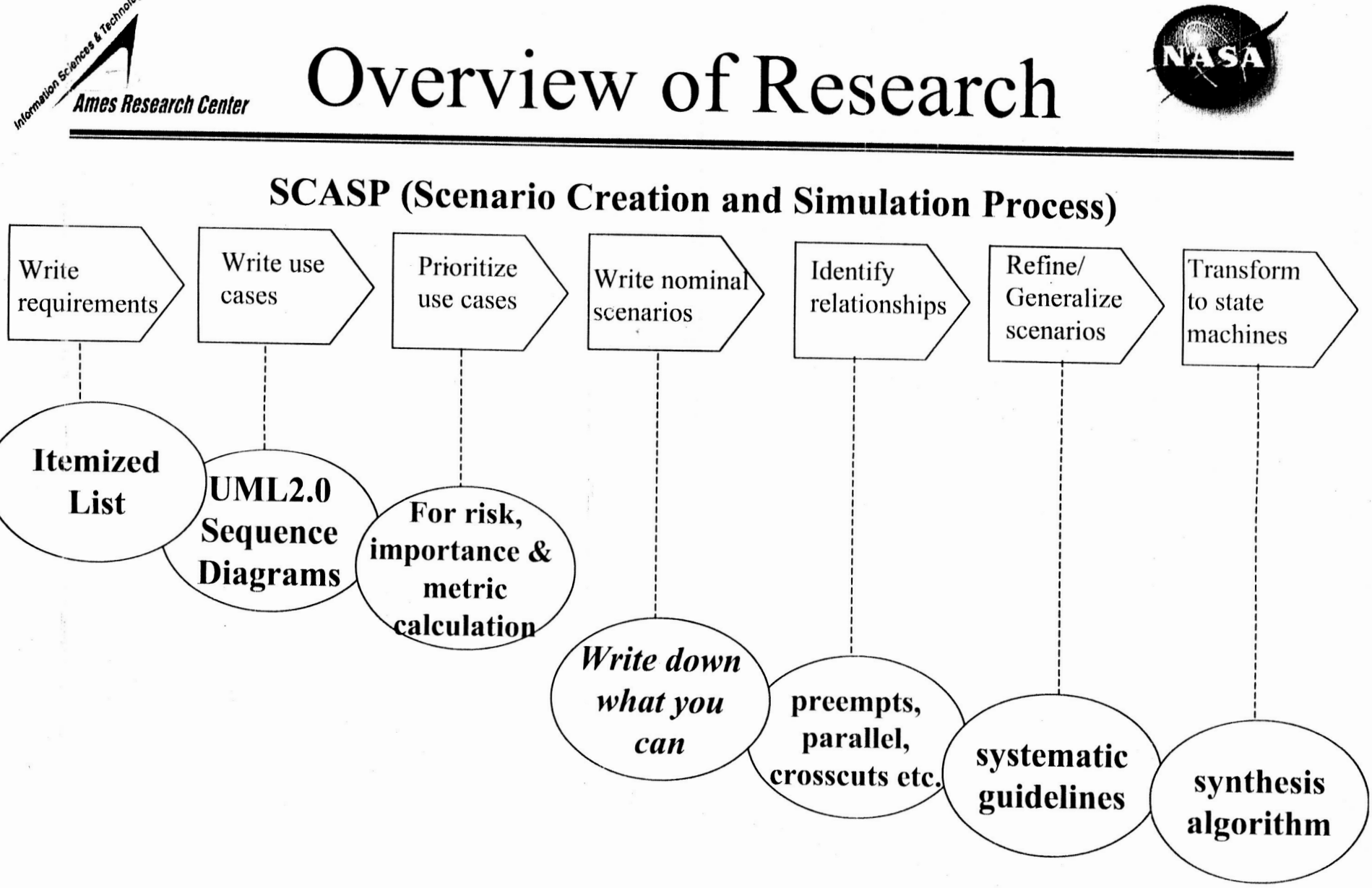


Goal:Cost-effective way of simulating requirements

- *automatic transformation to executable form*
- *executable form can be reused in design*

Information Sciences & Technology
Ames Research Center

Overview of Research

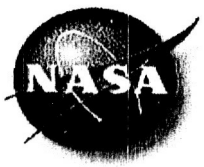


Importance/Benefits

- Thorough simulation of use cases *before* design/implementation
  - reduced cost
  - fewer misunderstandings
  - reuse of executable form of use cases
- SCASP gives *systematic guidelines* on how to:
  - separate concerns in use case descriptions
  - elicit non-nominal scenarios (alternatives, exceptions, concurrent scenarios etc.)
  - transform those scenarios automatically into a set of concurrent state machines
  - execute those state machines, i.e., scenario simulation
- **NASA relevance** (specific projects):
  - CTAS air traffic control (Ames)
    - weather update module
    - trajectory synthesizer
    - data link/uplink
  - also: Motorola test methodology (ENTITE)

Information Sciences & Technology
Ames Research Center

Accomplishments



- **SCASP:**
  - Defined SCASP and evaluated on multiple case studies
    - *CTAS weather update*
    - *Motorola call setup sequence*
    - *Univ. Paderborn shuttle system*
    - *CTAS trajectory up/downlink*
  - Techniques for separation of concerns (aspects)
    - *forthcoming papers in Requirements Engineering '04 and IEE Software*
  - Synthesis:
    - *outlined new algorithm for synthesizing state machines from scenarios*
  - Metrics
    - *defined metrics for evaluating completeness/complexity of process*
- **Tool support (IBM Rational Rose plug-in):**
  - Simple version of algorithm
  - plug-in for reusable patterns (including use case aspects)
  - Integrated state machine simulator from Teknowledge Corp. (Alexander Egyed)
- **Customer interest:**
  - NASA CTAS
  - Motorola

Next Steps

- **SCASP:**
  - Further evaluation on case studies
  - Synthesis:
    - *develop, implement and test new algorithm*
  - Metrics:
    - *evaluation*
  - Simulation:
    - *feedback simulation results*
- **Tool support:**
  - Full version of algorithm
  - Integration
- **Customer transfer**

Transitioning From Software Requirements Models to Design Models

PI: Jon Whittle

QSS Group Inc.

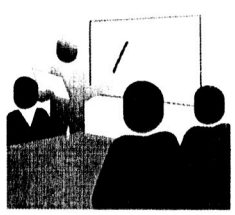
Information Sciences & Technology
Ames Research Center

Use case simulation

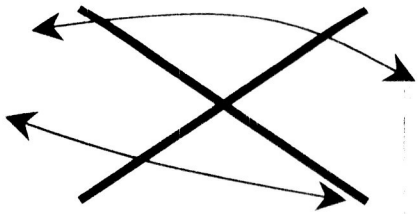
- Complexity
- Validation
- Evolution
- Requirements/Design gap

→ Simulation of use cases (scenarios)

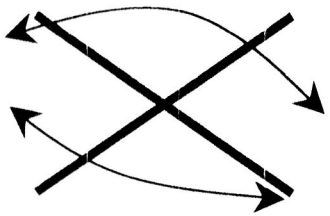
→ Semi-automatic design transformation



customers

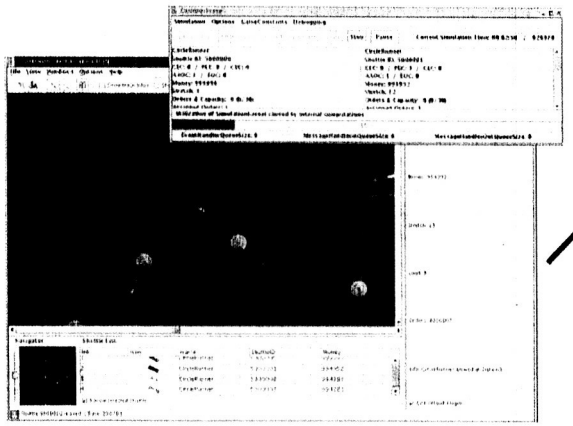


designer/architect



programmer

simulation

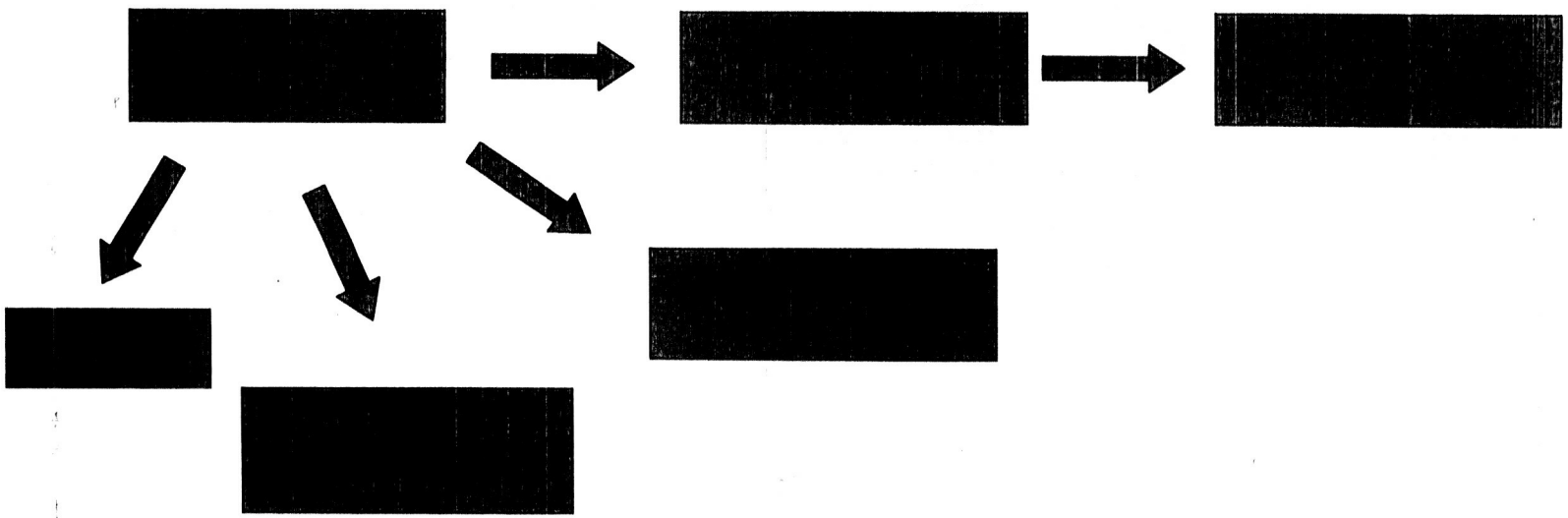


Goal: Cost-effective way of simulating requirements

- *automatic transformation to executable form*
- *executable form can be reused in design*

Information Sciences & Technology
Ames Research Center

Main Idea



Many good reasons for working with scenarios

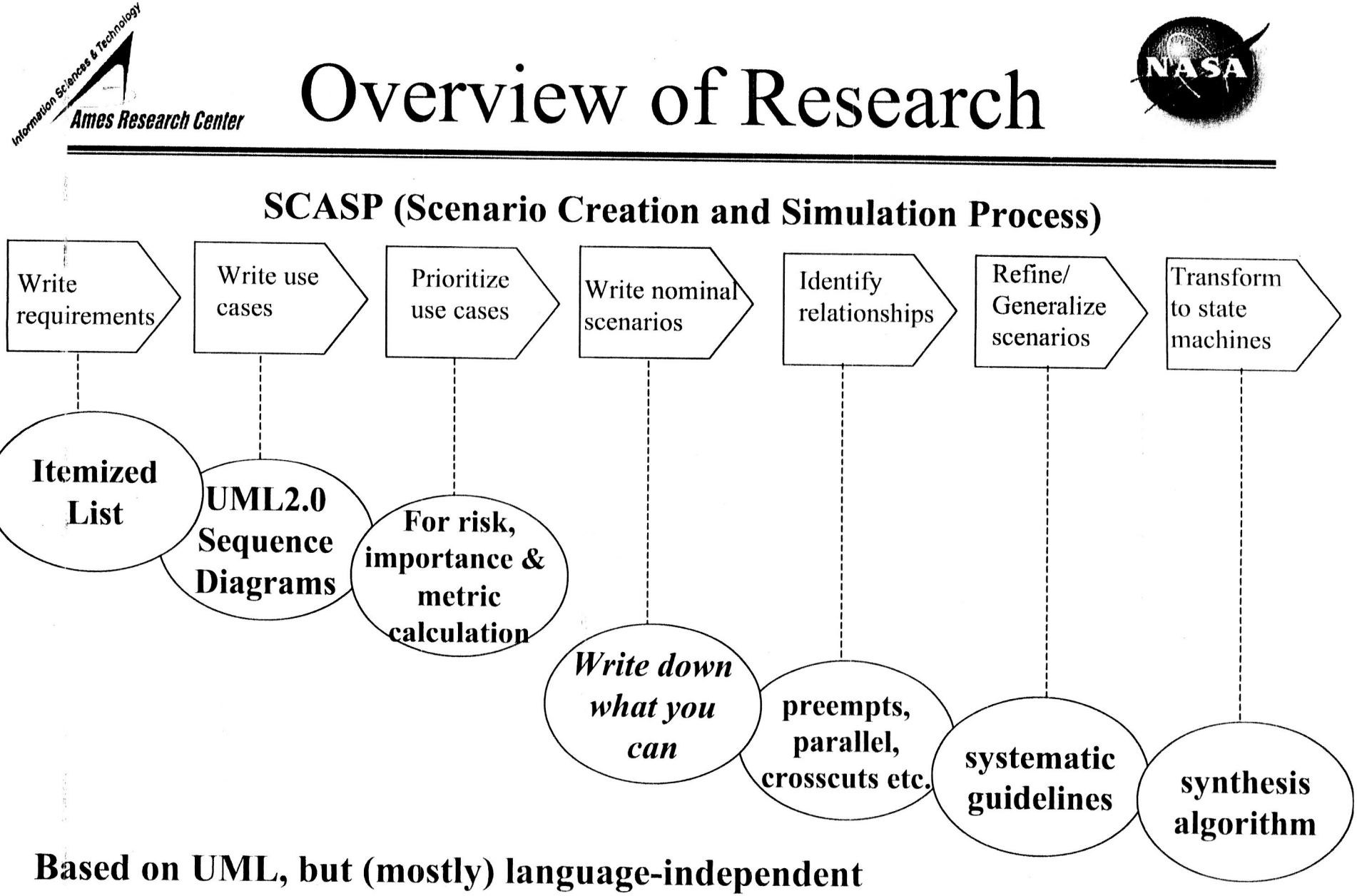
- walkthrough software artifacts
- analysis/validation
- test case generation
- state machine generation

Missing link: how to develop an appropriate set of scenarios?

- **synthesis requires *completeness***
- **test case generation requires *coverage***
- **requirements validation requires *coverage***

Information Sciences & Technology
Ames Research Center

Overview of Research

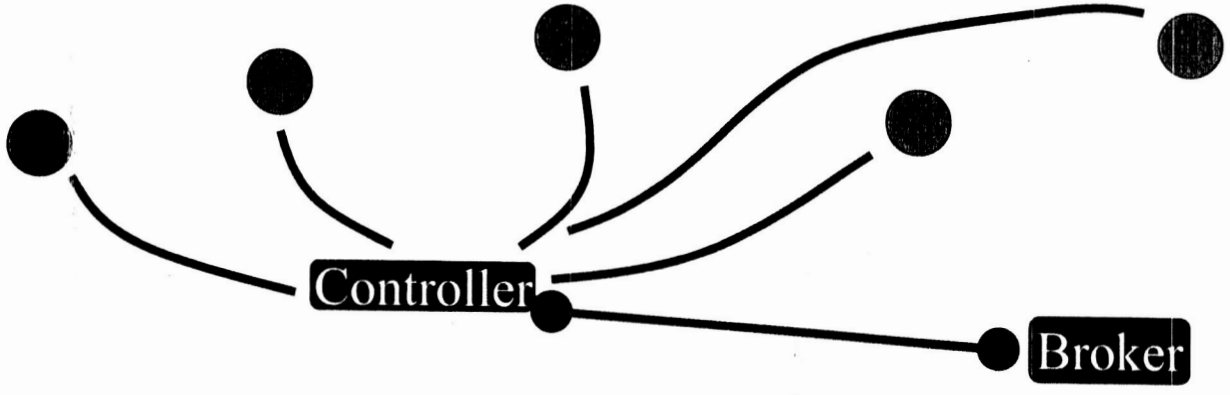


Based on UML, but (mostly) language-independent

Metrics measure completeness/complexity

Illustrative Example

- Autonomous **agents** bid for **orders** from **clients**
  - May bid for any number of orders simultaneously
  - Broker assigns orders
  - Clients pay controller who in turn pays agents



This talk:
agent=rail shuttle
client=passenger

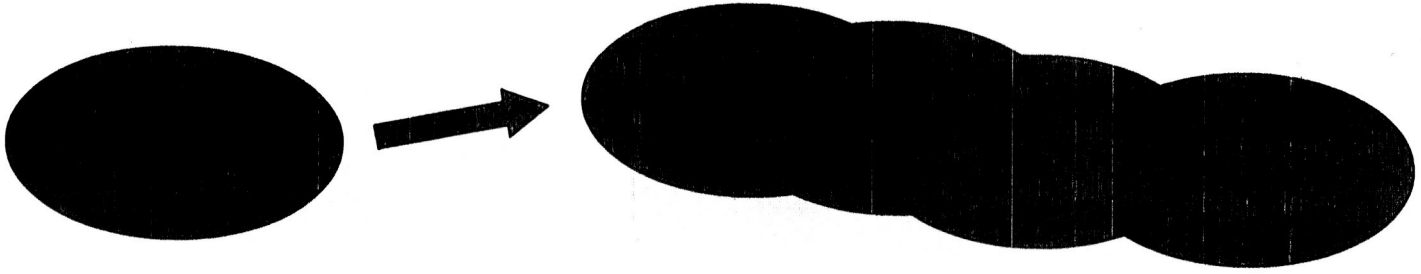
Information Sciences & Technology
Ames Research Center

1. Write Requirements

R1	Shuttles traveling on sections of tracks that are disrupted are not affected.
R2	Shuttles not traveling on a section of tracks that become disrupted will not be able to use it.
R3	All shuttles will be informed of a disruption and its duration.
R4a	Orders are made known to all shuttles by a broker.
R4b	Any shuttle can make an offer within a certain period of time
R4c	The shuttle making the lowest offer will receive the assignment
R4d	In the event of two equal offers, the assignment goes to the shuttle that made the first offer
R5	Orders will be paid for by passengers either by credit card or invoice
R6a	Every shuttle has a maximum capacity determined at the start of the simulation
R6b	A shuttle can transport more than one order at a time as long as the orders do not exceed the maximum capacity
R6c	The number of orders assigned (not necessarily loaded) to a shuttle at any given time is not limited
R7a	To complete an order, a shuttle has to travel to the start station, load the order and proceed to the destination station to unload
R7b	**R7a** has to be completed within a deadline or a penalty will be levied.
R7c	Loading/unloading at intermediate stations (for the same order) is not permitted
R8	A shuttle traveling on a section of tracks can neither change direction nor choose another destination. A travel decision is only possible at a station before the journey has begun.
R9a	In the beginning, every shuttle will receive a fixed capital
R9b	Payment to a shuttle occurs after an order is delivered and an invoice is sent to the banking agent
R9c	If the order specified credit card payment, money is transferred to the shuttle immediately. If the payment was invoicing then the transfer will be delayed for a certain amount of time
R10	Shuttles pay their toll when a station is reached
R11	If a shuttle exceeds a certain distance, maintenance will be carried out at the next station automatically and the shuttle will not be able to leave the station until maintenance is finished.

Information Sciences & Technology
Ames Research Center

4. Write nominal scenarios

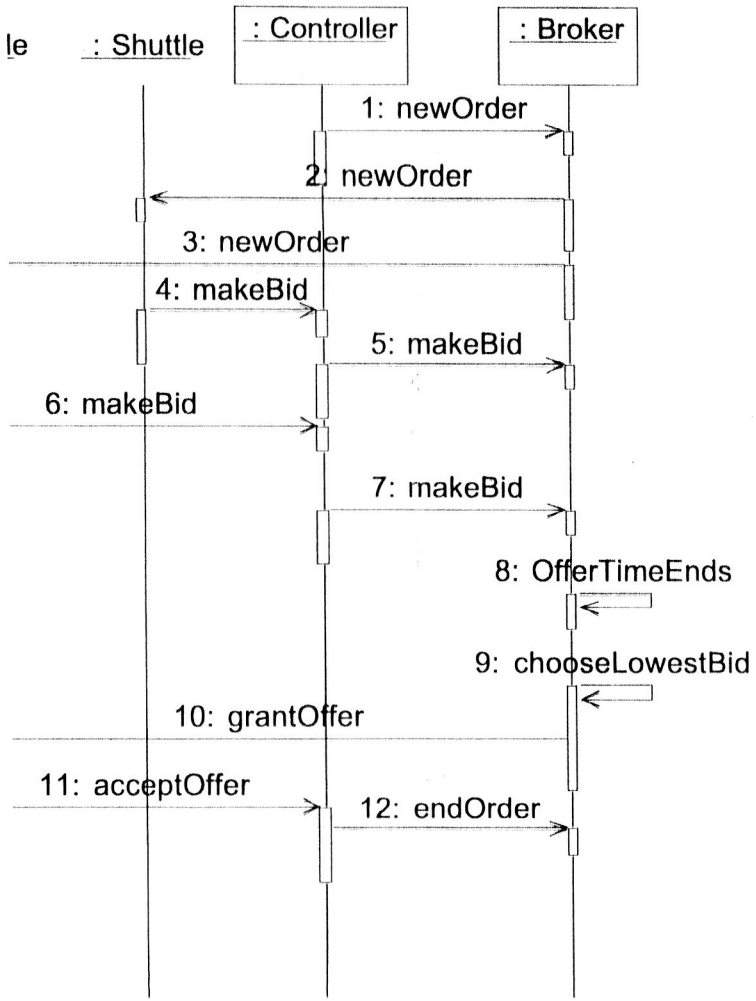


- *Nominal scenario:* typical or important functionality
- *Non-nominal scenario:* unusual or unexpected behavior

"write down what you can!"

Information Sciences & Technology
Ames Research Center

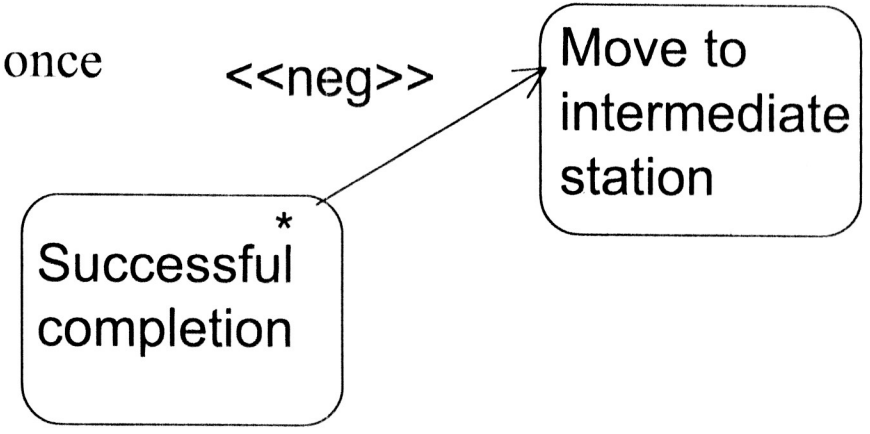
Nominal Scenario: bidding



5. Identify Relationships

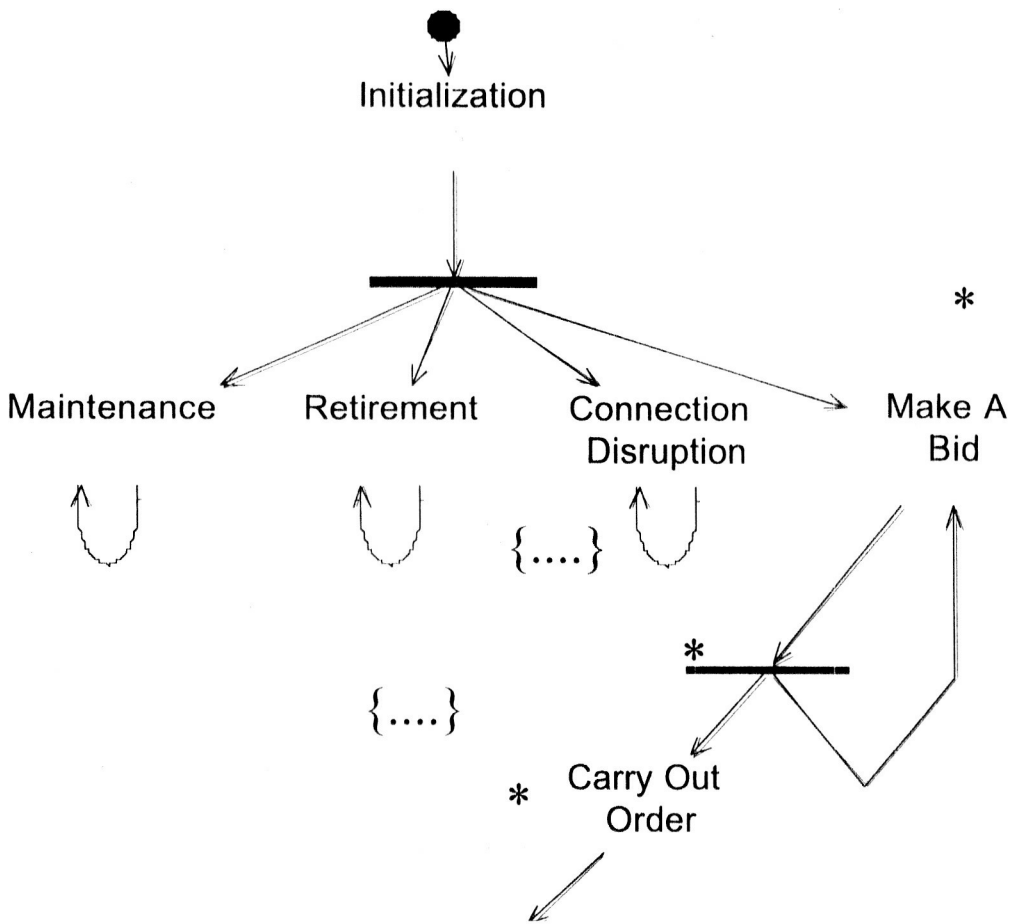
5.1 Within use cases:

S is a continuation of *T*
S is an alternative to *T*
S may execute in parallel with *T*
S may preempt *T*
S may suspend *T*
S may never happen during *R*
S is an exception handler for *T*
Multiple instances of *S* may execute at once
S crosscuts *R*.



5. Identify Relationships

5.2 Between use cases:



6. Generalize/Refine Nominal Scenarios

- Series of *issues* based on common generalization/refinement strategies.
  - May be language dependent or independent

Issue	**Context**	**Question**	**Action**	**Alternatives**
A generalization /refinement strategy to look for	Component, message or scenario	What to look for?	What action to take?	Alternative actions

Information Sciences & Technology
Ames Research Center

Issues (so far)

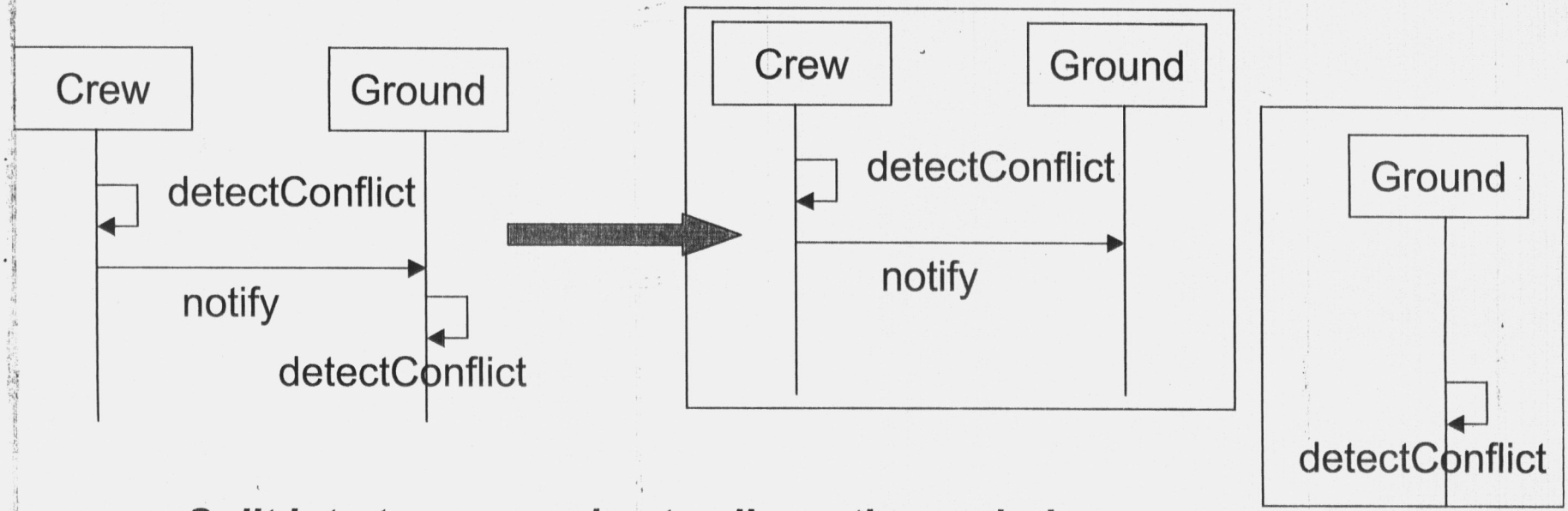
Context:		
1.1	Does the scenario apply to all	If no: consider a refactoring of the type model to introduce sub-
1.2	Should a message sent to *component* be sent to all	If all: replace *component* with a multiobject representing all components of type *type* and make
1.3	Should a message received by *component* be received by all	If all: replace *component* with a multiobject representing all components of type *type* and make
1.4	Analog of 1.2 for messages sent to *at least one* of a set of components of	
1.5	Analog of 1.3 for messages received by *at least one* of a set of components of a certain type.	

Context: *message*		
2.1	Could *message* be replaced with another message without changing the behavior of the scenario?	If yes: replace *message* with a combined fragment with interaction operator alt and the two alternative messages as operands.
2.4	Can the ordering of *message* be changed? In particular, could the ordering of *message* and its immediate neighbors be altered? Could the ordering of *message* and its distant neighbors be altered?	If yes: introduce coregions to relax the scenario ordering constraints.
		with operator **opt**.
2.6	Does *message* have a guard?	If yes: introduce alternatives when the guard is not satisfied (using **alt**).
2.7	Can *message* fail?	If yes: capture the failure handler as a separate interaction diagram referred to (using **ref**) in the main diagram and use an **alt** operator to capture the alternative when the message fails.
3.1	Are there undesired implied scenarios? – e.g., caused by messages on different lifelines that appear to be ordered based on their graphical depiction but are not ordered according to the sequence diagram semantics.	If yes: introduce an explicit "handshake" message between the two lifelines that forces the sequence diagram semantics to constrain the ordering of the messages.
	based on their graphical depiction but are not ordered according to the sequence diagram semantics.	ordering of the messages.

Example

2.11	Does *message* really depend on all its predecessors?	If no: extract the dependent messages into a separate scenario.

Scenario shows just *one* example, therefore factor out any "incidental" dependencies



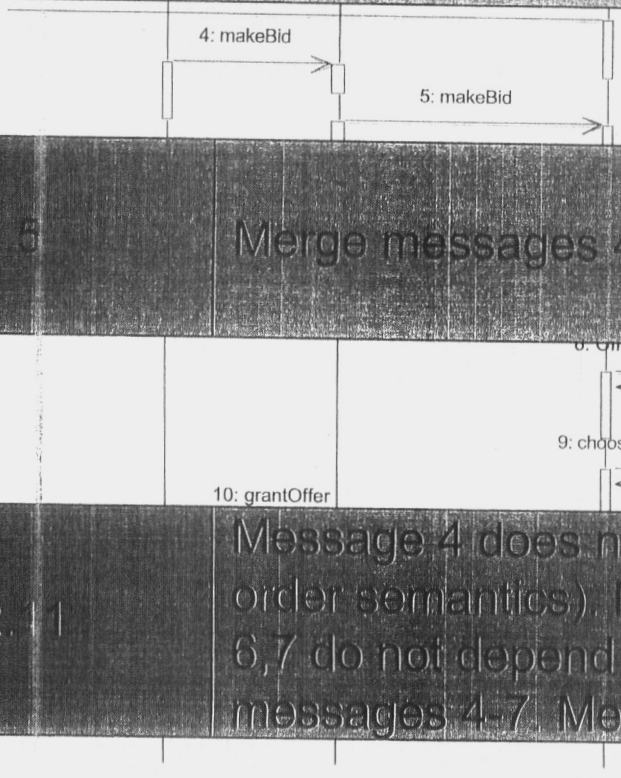
Split into two scenarios to allow other orderings

Information Sciences & Technology

Ames Research Center

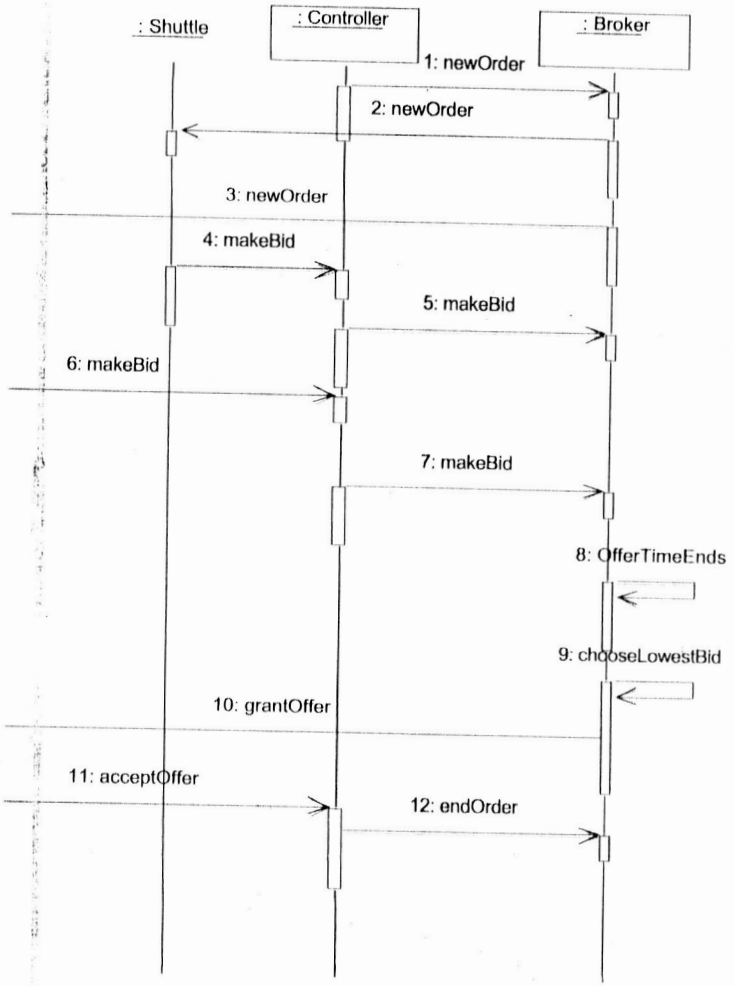
Bidding Example: Before

Issue	Description
1.2	Scenario shows only 2 shuttle instances. Generalize using a multiobject and merge messages 2 and 3 into a universal
1.2	Scenario shows only 2 shuttle instances. Generalize using a multiobject and merge messages 2 and 3 into a universal message. Note: there needs to be a single identified instance of Shuttle to receive message 10.
2.4	Messages 2,3 can be in any order. As can 4,6 and 5,7
2.5	Messages 4,5 can be marked as optional as only one shuttle may decide to bid
1.5	Merge messages 4 and 6 into an existential message
2.10	The order/bid for each shuttle can be split into parallel fragments
2.11	Message 4 does not depend on 3 (this one is ok because of the partial order semantics). Message 6 does not depend on
2.11	Message 4 does not depend on 3 (this one is ok because of the partial order semantics). Message 6 does not depend on message 2. Messages 6,7 do not depend on message 5. Message 8 does not depend on messages 4-7. Message 11 does not depend on messages 4 and 5.



Information Sciences & Technology
Ames Research Center

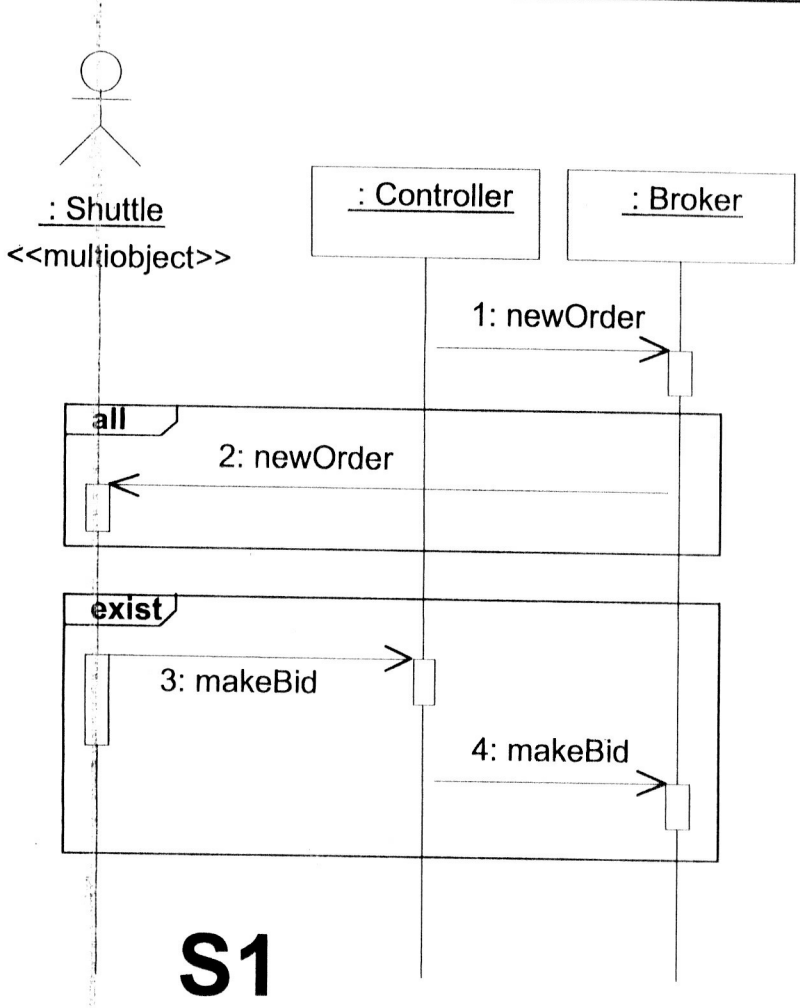
Bidding Example: Before



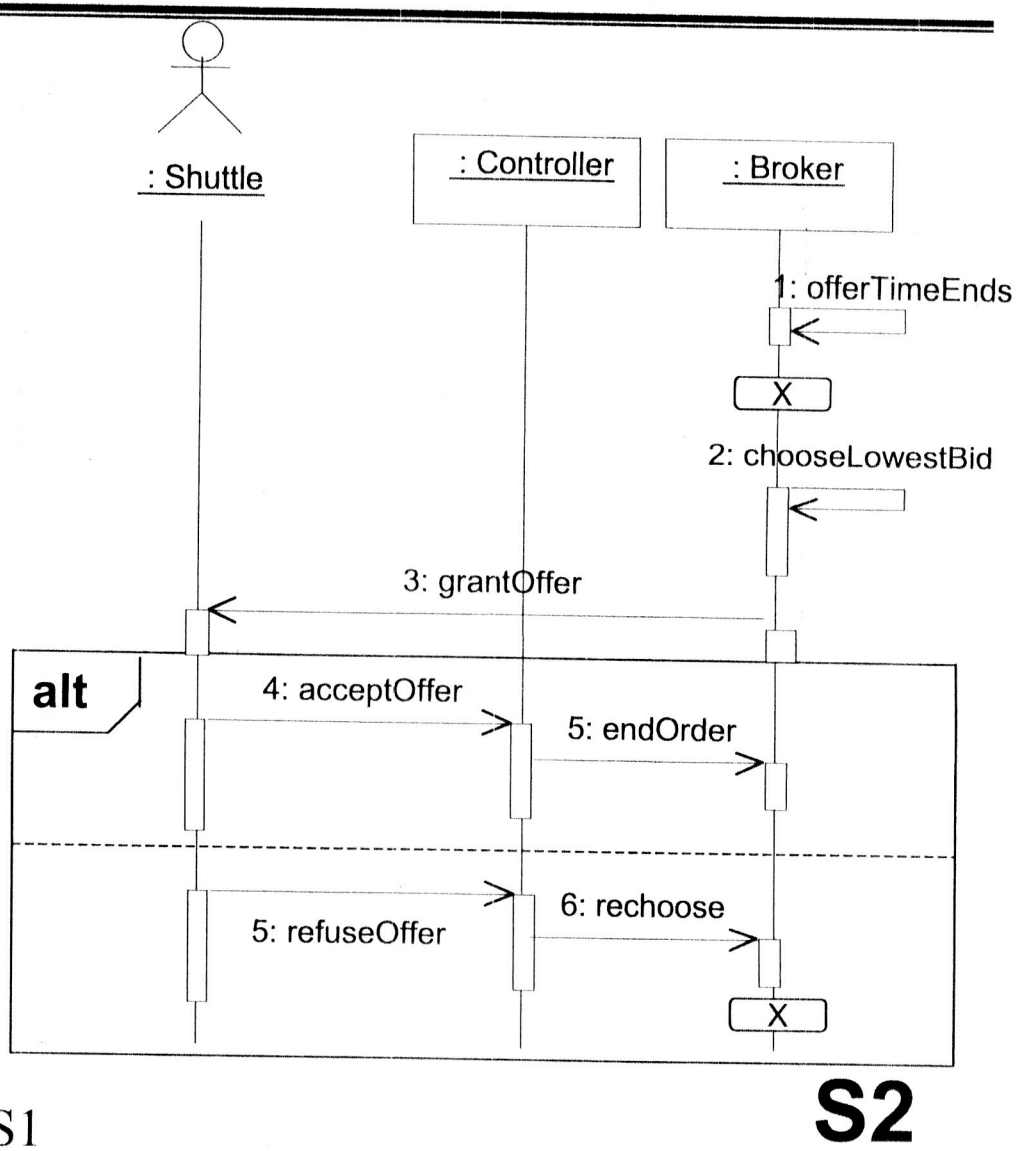
Issue	Description
1.2	Scenario shows only 2 shuttle instances. Generalize using a multiobject and merge messages 2 and 3 into a universal message. Note: there needs to be a single identified instance of Shuttle to receive message 10.
1.5	Merge messages 4 and 6 into an existential message
2.3	Alternative to message 11 is *rejectOffer*. Alternatives to 9 are *noBids* and *BidsAreEqual*.
2.4	Messages 2,3 can be in any order. As can 4,6 and 5,7
2.5	Messages 4,5 can be marked as optional as only one shuttle may decide to bid Message 9 is optional (there may be only one bid) Message 11 is optional as the winning shuttle may not respond to the offer
2.7	There may be communication failures
2.9	The broker should not accept the highest bid
2.10	The order/bid for each shuttle can be split into parallel fragments
2.11	Message 4 does not depend on 3 (this one is ok because of the partial order semantics). Message 6 does not depend on message 2. Messages 6,7 do not depend on message 5. Message 8 does not depend on messages 4-7. Message 11 does not depend on messages 4 and 5.

Information Sciences & Technology
Ames Research Center

Bidding Example: After



S1



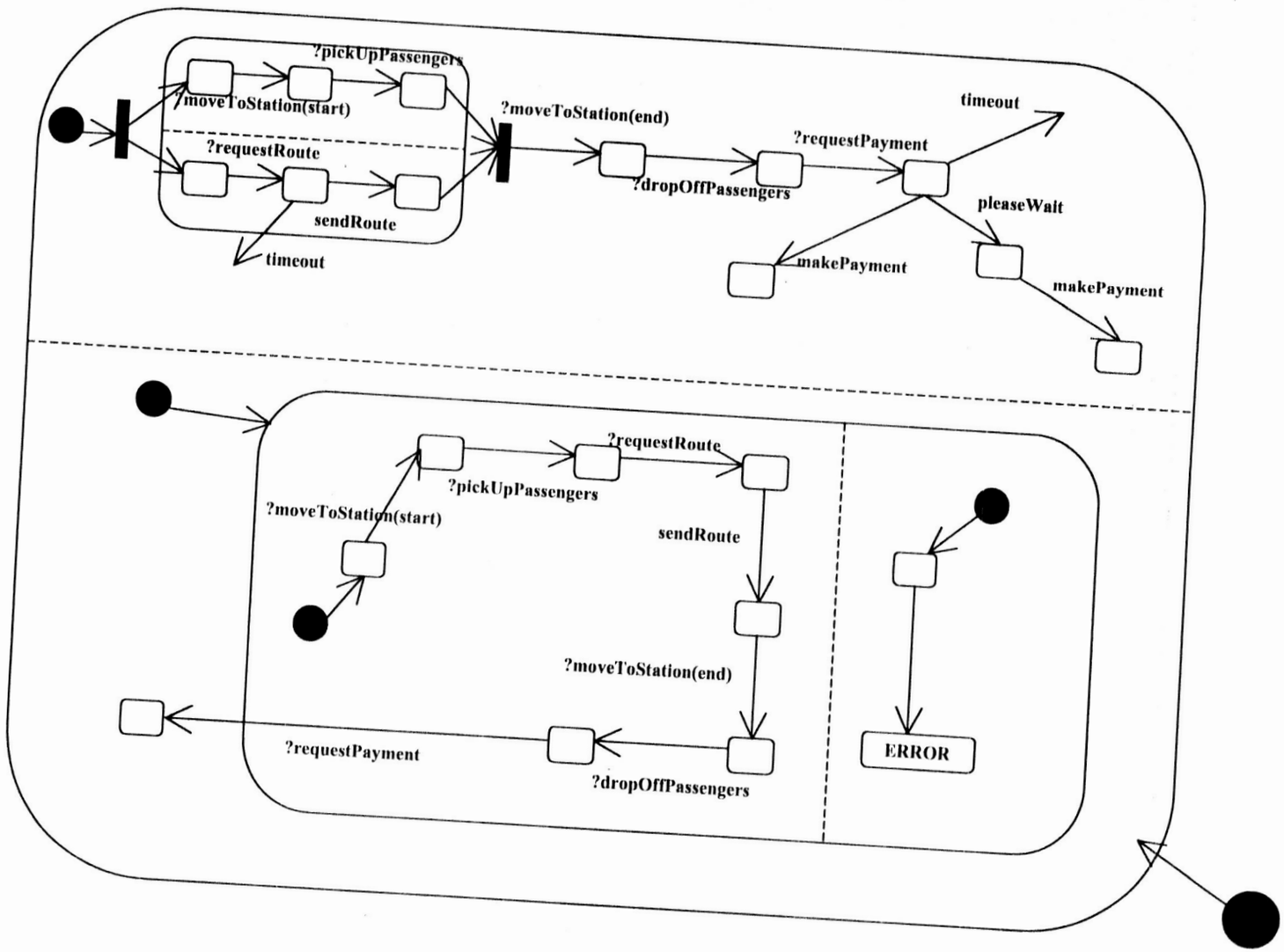
S2

S2 preempts S1

Information Sciences & Technology

Ames Research Center

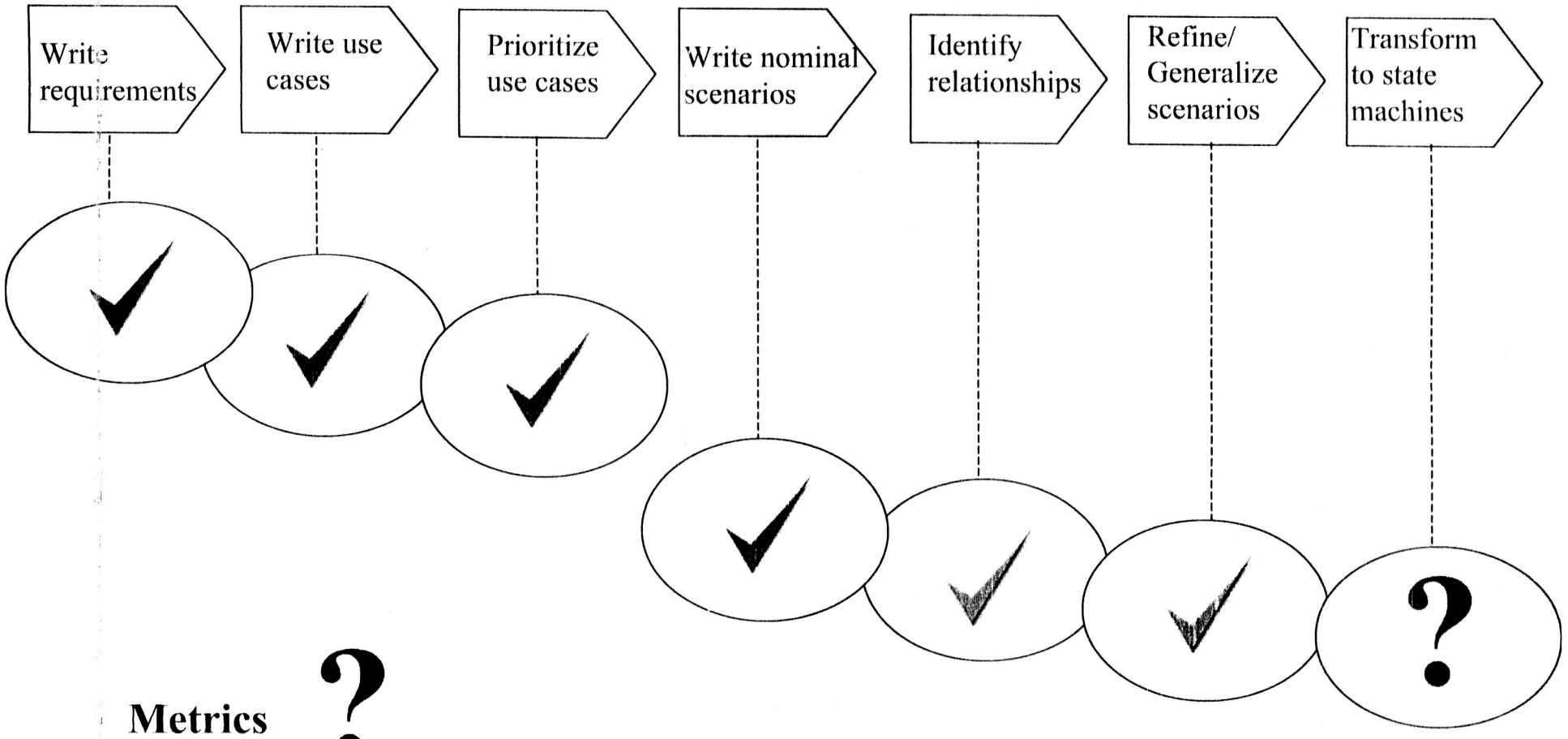
7. Transform to State machines



Information Sciences & Technology
Ames Research Center

"State of Play"

SCASP (Scenario Creation and Simulation Process)



Metrics ?

TRL 5

Metrics

$$UCCP = \#w \times \frac{NT_u / ECP_u}{\sum_u NT_u / ECP_u}$$

actual complexity

nonzero priority & nonzero complexity

nonzero priority

$$ECP_u = P_u \times EC_u$$

priority

expected complexity

Information Sciences & Technology
Ames Research Center

Accomplishments

- **SCASP:**
  - Defined SCASP and evaluated on multiple case studies
    - *CTAS weather update*
    - *Motorola call setup sequence*
    - *Univ. Paderborn shuttle system*
    - *CTAS trajectory up/downlink)*
  - Techniques for separation of concerns (aspects)
    - *forthcoming papers in Requirements Engineering '04 and IEE Software*
  - Synthesis:
    - *outlined new algorithm for synthesizing state machines from scenarios*
  - Metrics
    - *defined metrics for evaluating completeness/complexity of process*
- **Tool support (IBM Rational Rose plug-in):**
  - Simple version of algorithm
  - plug-in for reusable patterns (including use case aspects)
  - Integrated state machine simulator from Teknowledge Corp. (Alexander Egyed)
- **Customer interest:**
  - NASA CTAS
  - Motorola